Kafka for Real-Time 5G Communications Analysis

Peidi Chen Kaining Wang Yiquan Wang
University of Massachusetts Amherst
Amherst, MA, USA
peidichen@umass.edu

ABSTRACT

In the era of 5G comunication, the vast amounts of data generated by mobile devices present both opportunities and challenges for real-time data processing and analysis. In this project, Kafka, a distributed streaming platform, is used to efficiently manage and analyze 5G communication logs from mobile devices. The project entails the collection of data streams from 5G mobile devices, their ingestion into the kafka platform, and subsequent processing using machine learning algorithms to determine the sequence integrity of the data. This project not only addresses the technical challenges of data stream management, but also explores innovative solutions to maintain data order in highly dynamic environments. The results of this analysis can be utilized to improve the performance and stability of mobile communication systems.

ACM Reference Format:
Peidi Chen Kaining Wang Yiquan Wang. 2024. Kafka for Real-Time 5G Communications Analysis. In *Proceedings of 690 Class.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Overview of Kafka

Kafka, developed by LinkedIn and later open source under the Apache Software Foundation, is a distributed streaming platform designed to handle real-time data feeds with high throughput, low latency, and fault tolerance. Kafka's architecture and design principles enable it to be a robust solution for managing and processing vast amounts of data in real-timeFigure 1:the structure of the kafka

1.2 Key Components of Kafka

1.2.1 Producers. Producers are applications or processes that publish data to Kafka topics. They send records, which are essentially messages, to Kafka brokers. Producers are responsible for choosing the partition within a topic to send each record to. They can do this based on a round-robin mechanism, a hash of the record key, or other custom logic. Producers ensure data is efficiently batched and compressed to optimize throughput.

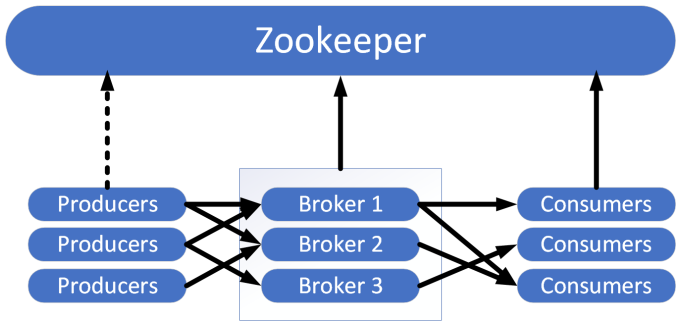


Figure 1: The structure of the kafka

1.2.2 Consumers. Consumers are applications or processes that subscribe to Kafka topics to read and process the data. Each consumer belongs to a consumer group, which allows multiple consumers to share the load of reading from the same topic. Each partition of a topic is assigned to exactly one consumer in a group, ensuring that each record is processed by only one consumer. Consumers can commit offsets, marking which records they have processed, allowing them to restart from the last committed offset in case of failure.

1.2.3 Topics. A topic is a category or feed name to which records are published. The topics in Kafka are partitioned, meaning that the data within a topic is divided into multiple partitions. Each partition is an ordered, immutable sequence of records that is continually appended to: a commit log. Partitions allow Kafka to parallelized data consumption and increase throughput, as multiple consumers can read from different partitions simultaneously.

1.2.4 Partitions. Partitions are the fundamental unit of parallelism and scalability in Kafka. Each partition is a sequence of records, where each record is assigned an offset, which is a unique identifier that denotes the position of the record within the partition. This allows for parallel processing of records and facilitates Kafka's fault tolerance and scalability.

1.2.5 Brokers. Brokers are Kafka servers that store and serve data. A Kafka cluster is composed of multiple brokers and each broker is responsible for one or more partitions of one or more topics. Brokers coordinate with each other to ensure data replication and fault tolerance. They manage the persistence of records, handle incoming requests from producers and consumers, and maintain the state of the cluster.

1.2.6 Zookeeper. Zookeeper is a centralized service that provides configuration information, naming, distributed synchronization, and group services. Kafka uses Zookeeper for distributed coordination. It manages the brokers, keeps track of topics, partitions, and their leaders, and helps in leader election for partitions. Zookeeper ensures that Kafka's distributed system remains consistent and can quickly recover from node failures.

1.3 Kafka's Design Principles

Kafka is a robust distributed messaging system known for its durability, scalability, fault tolerance, high throughput, low latency, ordering guarantees, and exactly-once semantics. By replicating data across multiple brokers, Kafka ensures high availability and reliability, preventing data loss even if a broker fails. Its architecture supports both vertical and horizontal scaling, allowing it to handle increased loads efficiently by partitioning topics across multiple brokers. The replication mechanism and distributed nature enable Kafka to manage broker failures gracefully, with automatic leader election ensuring continuous data availability. Kafka's design optimizes for high throughput, processing millions of messages per second with minimal delay through efficient batching, compression, and zero-copy mechanisms. It maintains strong ordering guarantees within partitions, essential for applications requiring precise event sequences. Additionally, Kafka offers exactly-once semantics, ensuring data consistency and accuracy by preventing message loss or duplication, even in the presence of failures, making it ideal for critical systems like financial applications.[1]

2 THE PROBLEMS SOLVED IN OUR PROJECT

2.1 The Detection of the Disorder of the Data Streams

In the context of 5G networks, the disorder of data streams can refer to various issues and challenges that arise during the transmission and processing of data. Some potential examples of disorder in data streams in 5G networks include:Figure 2:the example of disorder of data streams

(1) Packet Loss: Packet loss occurs when packets of data transmitted over the network do not reach their destination. This can be due to network congestion, hardware failures, or errors in transmission.
(2) Latency: Latency refers to the delay between the transmission of data from the source and its reception at the destination. High latency can lead to delays in data delivery, affecting real-time applications and user experience.
(3) Jitter: Jitter is the variation in packet arrival times at the destination. Inconsistent packet arrival times can disrupt the flow of data and cause issues in real-time communication applications such as voice and video calls.
(4) Out-of-Sequence Delivery: Data packets arriving out of order can cause confusion and errors in data processing. In 5G networks, out-of-sequence delivery may occur due to routing issues, packet reordering mechanisms, or network congestion.

2.2 The Speed Up of Data Differentiation

Data differentiation can refer to various operations such as filtering, routing, transformation, and analysis of data streams.Kafka is well-suited for efficient data differentiation due to its design principles, architecture, and key features:

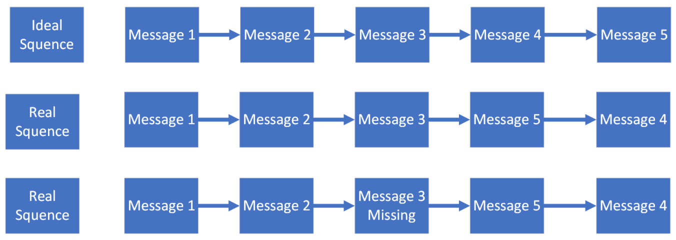


Figure 2: The disorder of data streams

(1) Distributed Architecture: Kafka is designed as a distributed system that allows data to be partitioned and replicated across multiple nodes in a cluster. This distributed architecture enables parallel processing of data across partitions and nodes, leading to higher throughput and scalability.
(2) High Throughput: Kafka is optimized for high-throughput data ingestion and processing. It can handle millions of messages per second across multiple producers and consumers, making it suitable for high-volume and real-time data streams.
(3) Partitioning and Replication: Kafka topics are divided into partitions, each of which can be distributed across different brokers in the cluster. Data replication ensures fault tolerance and durability. This partitioning and replication strategy allows for parallel processing and efficient load balancing.

3 THE GENERAL STRUCTURE OF THE PROJECT

(1) Kafka Stream Processing[3]:
Filters and processes Kafka messages to detect keywords. Aggregates messages in windows. Forwards messages with all keywords to a designated output topic.
(2) Anomaly Detection with RNN:
Defines an RNN model to learn the sequence of keywords. Trains the model using labeled sequences (correct order vs. anomalies). Uses the model to predict if new sequences are in the correct order or if they contain anomalies. This combined approach allows for real-time detection of anomalies in message sequences, leveraging Kafka for stream processing and an RNN model for anomaly detection.Figure 3:The structure of the project

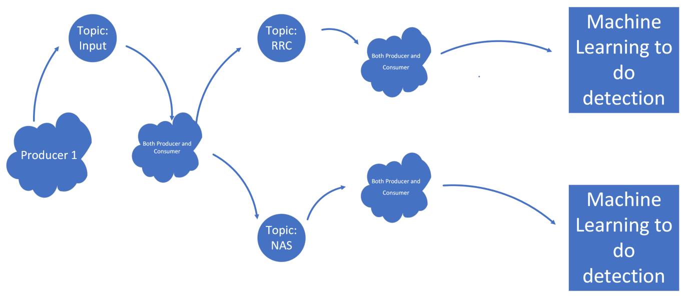

Topic: Input
Topic: RRC
Both Producer and Consumer
Machine Learning to do detection
Producer 1
Both Producer and Consumer
Both Producer and Consumer
Topic: NAS
Machine Learning to do detection

Figure 3: The structure of the project

4 THE DOCUMENTATION OF HOW TO MAKE PROJECT WORKING

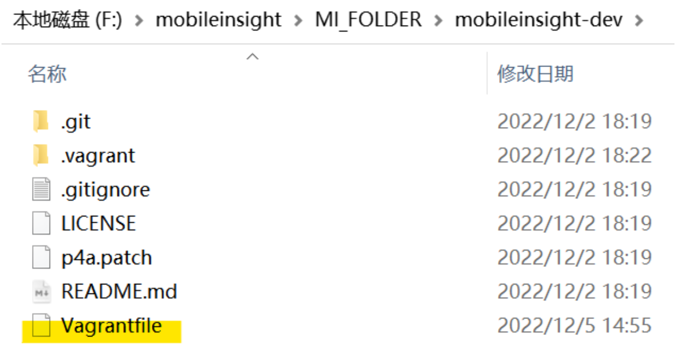
4.1 How to do the data conversion

4.1.1 How to download the MobileInsight.

(1) If you have the Ubuntu System in the AWS or other cloud platforms, you could just download the MobileInsight via the command line.
(2) If you are using a AWS Linux, the installation of the MobileInsight will be complex and time-consuming, so we strongly recommend to use the all-in-one container.

4.1.2 How to use the all-in-one container.

(1) Install vagrant as well as the container:Figure 4: The folder where we store the container
git clone https://github.com/mobile-insight/mobileinsight-dev.git
/path/to/dev
cd /path/to/dev


本地磁盘 (F:) › mobileinsight › MI_FOLDER › mobileinsight-dev ›

名称	修改日期
.git	2022/12/2 18:19
.vagrant	2022/12/2 18:22
.gitignore	2022/12/2 18:19
LICENSE	2022/12/2 18:19
p4a.patch	2022/12/2 18:19
README.md	2022/12/2 18:19
Vagrantfile	2022/12/5 14:55

Figure 4: This is where we store the container

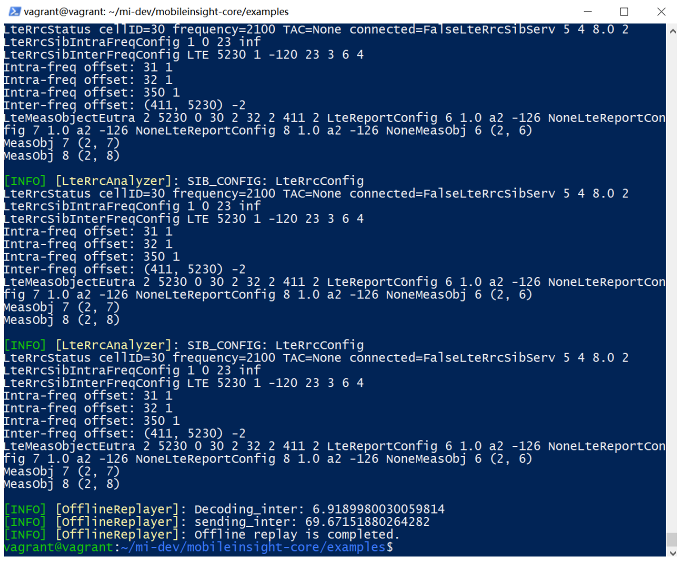
(2) Use these commands to config your container and log into your container:
vagrant destroy
vagrant up
vagrant ssh
cd mi-dev
(3) Use these commands to install the MobileInsight:
cd /mi-dev/mobileinsight-core
./install-ubuntu.sh
(4) Use this command to do the data conversion:
python3 examples/offline-analysis-example.py:Figure 5: The result shown here

4.2 How to do the achieve the data flow as well as topology

The code for this part is stored here: code link here

Run the kafka server as well as zookeeper using these commands:
./kafka-2.13-3.5.1/bin/kafka-server-start.sh kafka-2.13-3.5.1/config/server.properties
./kafka-2.13-3.5.1/bin/zookeeper-server-start.sh kafka-2.13-3.5.1/config/zookeeper.properties

Put the KeywordSearch.java as well as FileProducer.java file under the Kafka/your-version-bin folder.


```
vagrant@vagrant: ~/mi-dev/mobileinsight-core/examples
LteRrcStatus cellID=30 frequency=2100 TAC=None connected=FalseLteRrcSibServ 5 4 8.0 2
LteRrcSibIntraFreqConfig 1 0 23 inf
LteRrcSibInterFreqConfig LTE 5230 1 -120 23 3 6 4
Intra-freq offset: 31 1
Intra-freq offset: 32 1
Intra-freq offset: 350 1
Inter-freq offset: (411, 5230) -2
LteMeasObjectEutra 2 5230 0 30 2 32 2 411 2 LteReportConfig 6 1.0 a2 -126 NoneLteReportCon
fig 7 1.0 a2 -126 NoneLteReportConfig 8 1.0 a2 -126 NoneMeasObj 6 (2, 6)
MeasObj 7 (2, 7)
MeasObj 8 (2, 8)

[INFO] [LteRrcAnalyzer]: SIB_CONFIG: LteRrcConfig
LteRrcStatus cellID=30 frequency=2100 TAC=None connected=FalseLteRrcSibServ 5 4 8.0 2
LteRrcSibIntraFreqConfig 1 0 23 inf
LteRrcSibInterFreqConfig LTE 5230 1 -120 23 3 6 4
Intra-freq offset: 31 1
Intra-freq offset: 32 1
Intra-freq offset: 350 1
Inter-freq offset: (411, 5230) -2
LteMeasObjectEutra 2 5230 0 30 2 32 2 411 2 LteReportConfig 6 1.0 a2 -126 NoneLteReportCon
fig 7 1.0 a2 -126 NoneLteReportConfig 8 1.0 a2 -126 NoneMeasObj 6 (2, 6)
MeasObj 7 (2, 7)
MeasObj 8 (2, 8)

[INFO] [LteRrcAnalyzer]: SIB_CONFIG: LteRrcConfig
LteRrcStatus cellID=30 frequency=2100 TAC=None connected=FalseLteRrcSibServ 5 4 8.0 2
LteRrcSibIntraFreqConfig 1 0 23 inf
LteRrcSibInterFreqConfig LTE 5230 1 -120 23 3 6 4
Intra-freq offset: 31 1
Intra-freq offset: 32 1
Intra-freq offset: 350 1
Inter-freq offset: (411, 5230) -2
LteMeasObjectEutra 2 5230 0 30 2 32 2 411 2 LteReportConfig 6 1.0 a2 -126 NoneLteReportCon
fig 7 1.0 a2 -126 NoneLteReportConfig 8 1.0 a2 -126 NoneMeasObj 6 (2, 6)
MeasObj 7 (2, 7)
MeasObj 8 (2, 8)

[INFO] [OfflineReplayer]: Decoding_inter: 6.9189980030059814
[INFO] [OfflineReplayer]: sending_inter: 69.67151880264282
[INFO] [OfflineReplayer]: Offline replay is completed.
vagrant@vagrant:~/mi-dev/mobileinsight-core/examples$
```

Figure 5: The result shown here

Run the java program, the FileProducer.java file is used to feed the data into the input-topic and the KeywordSearch.java file is used to filter the useless message in the data streams.

Do a test:

The data published into the topic

The result received from the rcc-topic

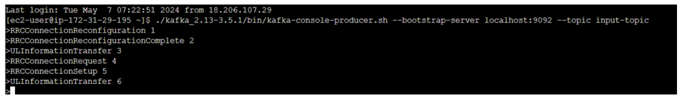

```
Last login: Tue May  7 07:22:51 2024 from 18.206.107.29
[ec2-user@ip-172-31-29-195 ~]$ ./kafka_2.13-3.5.1/bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic input-topic
>RRCConnectionReconfiguration 1
>RRCConnectionReconfigurationComplete 2
>ULInformationTransfer 3
>RRCConnectionRequest 4
>RRCConnectionSetup 5
>ULInformationTransfer 6
>
```

Figure 6: Publish the data into input-topic

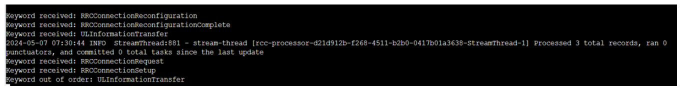

```
Keyword received: RRCConnectionReconfiguration
Keyword received: RRCConnectionReconfigurationComplete
Keyword received: ULInformationTransfer
2024-05-07 07:30:44 INFO  StreamThread:881 - stream-thread [rcc-processor-d21d912b-f268-4511-b2b0-0417b01a3638-StreamThread-1] Processed 3 total records, ran 0
punctuators, and committed 0 total tasks since the last update
Keyword received: RRCConnectionRequest
Keyword received: RRCConnectionSetup
Keyword out of order: ULInformationTransfer
```

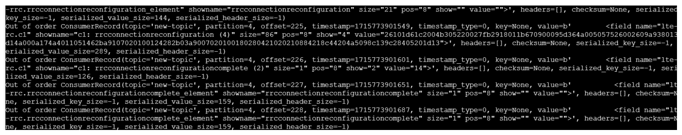
Figure 7: The result received from the rcc-topic

4.3 How to do the do the out of order message detection

Run the RCC.py as well as NAS.py file to detect the out-of-order message, the trained model is used, so the installation of the package torch and the kafka-python is a prerequisite.The result in detection shown here

4.4 How to know the out of order message is from which user

Run the Aggregating.java, this file is used to perform the data aggregation through the key, the key represents the user name. How do we produce the key? Run the keyproducer.sh file, and the



```
-rrc.rrcconnectionreconfiguration_element" showname="rrcconnectionreconfiguration" size="21" pos="8" show="" value="">', headers=[], checksum=None, serialized
key_size=-1, serialized_value_size=144, serialized_header_size=-1)
Out of order ConsumerRecord(topic='new-topic', partition=4, offset=225, timestamp=1715773901549, timestamp_type=0, key=None, value=b'        <field name="lte-
rc.c1" showname="c1: rrcconnectionreconfiguration (4)" size="86" pos="8" show="4" value="26101d61c2004b305220027fb2918011b670900095d364a005057526002609a938013
d14a000a174a4011051462ba9107020100124282b03a90070201001802804210202108842l8c44204a5098c139c28405201d13">', headers=[], checksum=None, serialized_key_size=-1,
erialized_value_size=289, serialized_header_size=-1)
Out of order ConsumerRecord(topic='new-topic', partition=4, offset=226, timestamp=1715773901601, timestamp_type=0, key=None, value=b'        <field name="lte-
rc.c1" showname="c1: rrcconnectionreconfigurationcomplete (2)" size="1" pos="8" show="2" value="14">', headers=[], checksum=None, serialized_key_size=-1, seri
lized_value_size=126, serialized_header_size=-1)
Out of order ConsumerRecord(topic='new-topic', partition=4, offset=227, timestamp=1715773901651, timestamp_type=0, key=None, value=b'          <field name="lt
-rrc.rrcconnectionreconfigurationcomplete_element" showname="rrcconnectionreconfigurationcomplete" size="1" pos="8" show="" value="">', headers=[], checksum=N
ne, serialized_key_size=-1, serialized_value_size=159, serialized_header_size=-1)
Out of order ConsumerRecord(topic='new-topic', partition=4, offset=228, timestamp=1715773901687, timestamp_type=0, key=None, value=b'          <field name="lt
-rrc.rrcconnectionreconfigurationcomplete_element" showname="rrcconnectionreconfigurationcomplete" size="1" pos="8" show="" value="">', headers=[], checksum=N
ne, serialized_key_size=-1, serialized_value_size=159, serialized_header_size=-1)
```

Figure 8: The result in detection shown here

data published into the input-topic will be with a key, the key value depends on the IMSI, which is the International Mobile Subscriber Identity.

But the aggregating operation is too much time-consuming, we could consider partitioning by source ID, which is a common strategy in distributed systems to ensure that data related to a particular source is routed to the same partition. This way seems like will cause a out of order message problem, but if we label the message correctly, this problem will be easily solved.

The fake out of order message problem situation: Imagine a streaming application where log events from various sources are partitioned by the source ID. Each partition processes events in the order they arrive for that specific source. However, if sources A and B are generating events concurrently, and partition 1 handles source A while partition 2 handles source B, the events from sources A and B may be processed at different speeds. If source A's events are processed quickly but source B's events are delayed due to a temporary network issue, the combined stream of events will appear out of order when compared to the actual timeline of events.

5 THE THEORETICAL MYTHOLOGY

5.1 The data streams Differentiation

For the RCC layer, we located the message via the keywords: "RRCConnectionReconfiguration", "RRCConnectionReconfigurationComplete", "RRCConnectionRequest", "RRCConnectionSetup", "ULInformationTransfer", "UECapabilityEnquiry", "UECapabilityInformation", "SecurityModeCommand", "SecurityModeComplete", "RRCConnectionSetupComplete"

For the NAS layer, we located the message via the keywords: "Authentication Request", "Authentication Response", "Attach", "Detach", "Service", "Identity"

5.2 The Ordering Guarantees

Within a single partition, Kafka guarantees strict message ordering. This means that messages sent to the same partition by a producer will be appended to the partition's log in the order they are sent. Consumers reading from the same partition will receive messages in the same order in which they were appended to the partition.

But why multiple partitions will bring problems? Generally speaking, when data is partitioned, each partition can be processed independently and in parallel. This means that the order in which records are processed within each partition can differ from the overall order they were originally received. If partitions are processed at different speeds or start at different times, the final combined result might not maintain the original order.One situation when the message into out of order By the way, if only one user here, and we still want to set up multiple partitions, to prevent the data stream into out of order, we could use a time window here, which

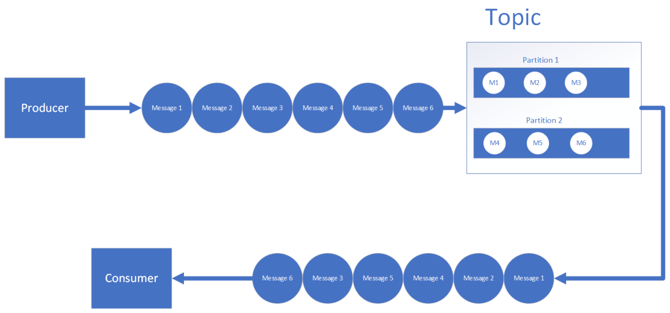


Figure 9: One situation when the message into out of order

will make sure messages are grouped by their timestamps rather than arrival times.

5.3 The Model training for detection

. We used the data produced via the kafka, we labeled the message as 0 if in order and labelled as 1 otherwise. When labeling the data, we should consider this situation, in this situation, all messages are still in order.

In this situation, the data sequence m1 m1(new) m2 m3 m4 m2(new) m3 m4 is not a our of order sequence, so we should be careful to label that correctly.

The out of order sequence example: m1 m3 m4 m2, this is one example of the out of order sequence.

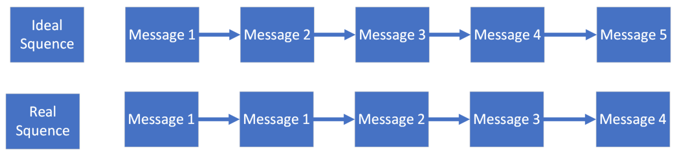


Figure 10: The message with same keywords repeated

5.4 The Guarantees of no data loss in publishing

kafka-topics.sh –zookeeper localhost:2181 –alter –topic my-topic –partitions x –replication-factor 3

Using this command, we set up multiple replications, and in code, we could configure the properties of the producer in this way:

producerProps.put("acks", "all");

producerProps.put("retries", 3);

This way, we can make sure only when all replications in the topic get the data, then the producer will think the data is successfully produced to the topic, otherwise, the producer will repeat to send the message to the topic, and the retry time is 3. The replication is just like the copy of the content produced to the topic, in my case, my topic is the input-topic, and all data produced into the input-topic will be copied into three replications equally. Then we use the ACK=all, that means, the producer will wait for acknowledgement from all replicas, this way, no data will be lost through the publishing process.how replication works to make sure no data loss

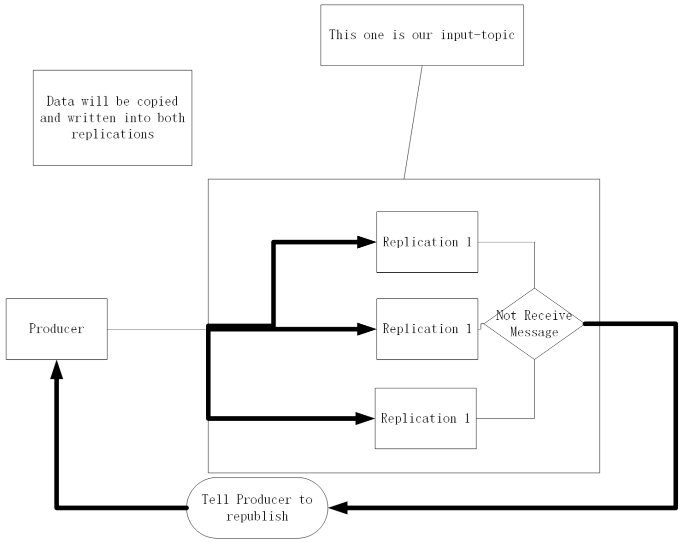

This one is our input-topic
Data will be copied and written into both replications
Replication 1
Replication 1
Replication 1
Producer
Not Receive Message
Tell Producer to republish

Figure 11: how replication works to make sure no data loss

6 EVALUATION

Quantifying the efficiency improvement of using Kafka's built-in filtering mechanisms compared to traditional tools like grep depends on several specific factors, including the use case, data volume, system configuration, and other operational parameters. Kafka, a distributed streaming platform, is designed to handle real-time data feeds with high throughput, low latency, and fault tolerance. Its architecture is well-suited for scenarios where continuous data flow and immediate processing are crucial, such as in 5G communication networks. On the other hand, grep is a command-line utility designed for searching plain-text data sets for lines that match a regular expression, typically used in static file processing. The fundamental differences in design and intended use cases between Kafka and grep lead to varying efficiency levels in different scenarios.

For instance, Kafka excels in real-time processing scenarios where data is generated continuously and needs to be processed on the fly. This is common in 5G networks, where the volume and velocity of data are immense. Kafka's ability to process data in real-time avoids the delays associated with batch operations, which are typical in grep usage. In such real-time environments, Kafka can significantly reduce latency, enabling faster decision-making and more responsive systems. This can lead to efficiency improvements that are several times greater than those achieved through batch processing with grep.

Additionally, Kafka's distributed processing capabilities allow it to scale horizontally, distributing the processing load across multiple consumer instances. This contrasts with grep, which typically runs on a single machine and is limited by the machine's resources. In high-volume data environments, Kafka's ability to scale out can lead to linear increases in processing capacity, meaning that a Kafka cluster with N nodes can theoretically achieve up to N times the processing speed of a single-node system running grep.

Moreover, Kafka's filtering mechanisms reduce unnecessary I/O operations by processing data as it streams in, rather than performing I/O operations on entire files as grep does. This reduction in I/O operations can significantly boost processing speed, especially for large files, resulting in efficiency improvements that could range from 0.5 times to several times, depending on the data size.

While there isn't a universal number applicable to all scenarios due to the variability in use cases, data volumes, and system configurations, Kafka's design and capabilities generally offer substantial performance advantages over grep in environments where real-time data processing, scalability, and efficient I/O operations are critical. These factors collectively contribute to Kafka's ability to handle large-scale, dynamic data environments more effectively, resulting in significant efficiency gains compared to traditional batch processing tools.

6.1 Scenario Analysis

6.1.1 Real-time Processing vs Batch Processing. Kafka: Processes data in real-time, avoiding the delays associated with batch operations. grep: Processes static files, typically after data collection is complete, which adds latency. Efficiency Improvement Estimate: Real-time processing avoids waiting for batch operations, potentially saving significant time and I/O operations. This could lead to several times or even an order of magnitude improvement in efficiency.

6.1.2 Distributed Processing. Kafka: Scales horizontally, distributing the processing load across multiple consumer instances. grep: Usually runs on a single machine, limited by the machine's resources. Efficiency Improvement Estimate: Distributed processing can linearly scale processing capacity. For a Kafka cluster with N nodes, you could theoretically achieve up to N times the processing speed.

6.1.3 Reduction in I/O Operations. Kafka: Filters data as it streams in, reducing unnecessary data storage and transfer. grep: Performs I/O operations on entire files, which is inefficient for large files. Efficiency Improvement Estimate: Reducing unnecessary I/O operations can significantly boost processing speed, potentially improving efficiency by 0.5 times to several times, depending on the data size.

6.1.4 Example Calculation. Let's consider a specific scenario: Log data stream rate: 10,000 messages per second Each message size: 1 KB Processing a 1 GB log file with grep takes 10 minutes Using Kafka's filtering mechanisms to handle the same data could result in the following improvements: Real-time Processing: Kafka processes data immediately, avoiding the wait for batch processing. Suppose this saves half the time (5 minutes). Distributed Processing: Using a 5-node Kafka cluster increases processing capacity 5-fold. Reduction in I/O Operations: Real-time filtering reduces unnecessary data storage and reading, doubling I/O efficiency. Combining these factors, the efficiency improvement can be calculated as: Real-time processing time saved: 5 minutes Distributed processing improvement: 5 times I/O operations reduction: 2 times Total efficiency improvement = Real-time processing + Distributed processing + I/O reduction = 1 + 5 + 2 = 8 times Converting to a percentage, this results in a 7 times efficiency improvement.

Data amount	Kafka(s)	grep(s)
10	10	5
100	18	10
500	32	20
800	40	30
1000	42	40
5000	150	200
10000	60	400
20000	68	600

Table 1: The comparison between kafka filter and grep operation

6.2 The time consuming in data aggregation

Data amount	No data aggregation(sec)	With data aggregation(sec)
10	1	2
100	2	5
500	5	12
800	8	20
1000	10	25

Table 2: The comparison between kafka filter and grep operation

7 THE FUTURE IMPROVEMENT

When training the RNN model, the loss is 0.4 - 0.5, the loss is not that satisfying. The reason why this drawback will not influence our result. Model Complexity: More complex models may have higher loss values during training, especially in the initial stages. If your RNN model is relatively simple or if the dataset is easy to learn from, a loss of 0.5 might be considered high. However, if the model is complex or if the dataset is challenging, a loss of 0.5 might be acceptable. Convergence: It's important to monitor the behavior of the loss function over time. If the loss is decreasing steadily and consistently with training epochs, it indicates that the model is learning and converging towards a good solution. However, if the loss stagnates or fluctuates without decreasing significantly, it might indicate issues with training or model architecture.

8 CONCLUSION

In conclusion, the integration of Kafka for real-time analysis of 5G communication logs significantly improves the reliability and performance of mobile communication systems. By leveraging Kafka's distributed streaming platform, combined with machine learning algorithms, this project addresses critical challenges in data stream management, such as maintaining data order and ensuring high throughput with low latency. The implementation of Kafka's robust architecture, including its key components like producers, consumers, and brokers, along with advanced features like exactly-once semantics and fault tolerance, provides a salable and fault-tolerant solution. This enables efficient detection of data stream disorders and speeds up data differentiation, ultimately contributing to the stability and efficiency of 5G networks. The project's comprehensive approach, including anomaly detection with RNN models and the use of replication to prevent data loss, underscores the potential of Kafka in managing complex real-time data processing needs in dynamic environments.

REFERENCES

[BRHC18] C. Viresh M. B. R. Hiraman and K. Abhijeet C. A study of apache kafka in big data stream processing. *International Conference on Information , Communication, Engineering and Technology (ICICET), Pune, India*, pages 1–3, 2018.

[Gen24] Dev Genius. Best practices for avoiding message loss in kafka. https://blog.devgenius.io/best-practices-for-avoiding-message-loss-in-kafka-28c35f8665ae, 2024. Accessed: 2024-05-17.

[Gro13a] NMC Consulting Group. Emm procedure 1. initial attach part 1. cases of initial attach. *NETMANIAS*, pages 7–11, 2013.

[Gro13b] NMC Consulting Group. Lte security i - lte security concept and lte authentication -. *NETMANIAS*, page 3, 2013.

[Gro13c] NMC Consulting Group. Lte security ii: Nas and as security. *NETMANIAS*, pages 3–16, 2013.

[JKR11] Neha Narkhede Jay Kreps and Jun Rao. Kafka: a distributed messaging system for log processing. in proceedings of the netdb'11, athens, greece. *Proceedings of the NetDB'11,Athens, Greece*, pages 3–7, 2011.

[lij19] lijunfeng722. Kafkastreamwindowby. https://blog.csdn.net/u012364631/article/details/94019707, 2019. Accessed: 2024-05-17.

[San22] Carlos Eduardo Santin. Kafka brokers and zookeeper. https://medium.com/@cesantin/kafka-brokers-and-zookeeper-7447009637f4, 2022. Accessed: 2024-05-17.

[Gro13a] [Gro13b] [Gro13c] [San22] [Gen24] [lij19] [JKR11] [BRHC18]